Software Engineering Group Project 20
Maintenance Manual

Author:	Kain Bryan-Jones[kab74],
	Luke Wybar[law39],
	Tom Perry[top19]

Config Ref:	MaintenanceManualGroup20
Date:	28th April 2020
Version:	0.1
Status:	Review

Department of Computer Science
Aberystwyth University
Aberystwyth
Ceredigion
SY23 3DB

CONTENTS

1. INTRODUCTION

Clear, consistently followed, document standards are essential in software engineering.

1.1 Purpose of this Document

This document describes an in-depth view of the Welsh Vocabulary Tutor program. This document should be read by any developers or 'maintainers' who are looking for specific answers to questions they have regarding how the Welsh Vocabulary Tutor program works. It will be useful for the user to be familiar with the Design Specification Documentation [1] as it will be a reference a lot in this document. This is to save time in this document where the Design Specification has already provided a sufficient description.

1.2 Scope

This document describes an in-depth view of the Welsh Vocabulary Tutor program. This document should be read by any developers or 'maintainers' who are looking for specific answers to questions they have regarding how the Welsh Vocabulary Tutor program works. It will be useful for the user to be familiar with the Design Specification Documentation [1] as it will be a reference a lot in this document. This is to save time in this document where the Design Specification has already provided a sufficient description.

1.3 Objectives

The objective of this document is to:

Provide maintainers with the resources they need to understand the workings of the Welsh Vocabulary Tutor program.

Provide maintainers with a detailed description of the program so that they can use it to solve any issues they may be encountering.

Provide maintainers with the knowledge on how to safely implement features into the program. I.e So that the maintainer does not break the current working code.

2. PROGRAM DESCRIPTION

Welsh Vocabulary Tutor provides a user interface where users can practice their Welsh ability. Like a standard dictionary template, WVT provides a dictionary entry. Dictionary entries consist of a Welsh word and an English word, where both words have the same meaning. It also consists of a boolean value of whether the word itself is a practice word. That is, has the user marked said work to practice?

WVT works with JSON to store dictionary information. The user will need to provide the JSON file to be loaded/saved from/to. It is recommended that the JSON filenames are named appropriately such as 'dictionary.json'.

3. PROGRAM STRUCTURE

The Design Specification document [1] outlines in detail the program structure. For the following pieces of information please see the referenced section of the document.

Section 2.1 provides a description of the different functionalities provided by the Welsh Vocabulary Tutor program. A list of all modules(packages) is provided along with a description of the purpose of each module.

Section 2.2 provides a more detailed look into each of the modules of the program. Every class in the modules is displayed along with a description of what each class provides.
Section 4 provides a list of all methods in each of the classes. These classes are grouped by which module they lie in. This section also provides a description of each method as well as the signature of that method, i.e the name of the method, its return type, its access modifier, as well as the parameters it takes as input.

4. ALGORITHMS

The Design Specification document [1] outlines the significant algorithms within the program. Descriptions of these algorithms are also provided within the document as well as when are where these algorithms take place. What must happen before this algorithm run and what happens as a result of said algorithms running.

Section 5.2 provides a list of all significant algorithms as well as a description of those algorithms

5. THE MAIN DATA AREAS

The Design Specification document [1] details where important information is stored in the program. The main area of data is the storage of DictionaryEntry objects within a linked list. This is done within the Application Class in the package 'uk.ac.aber.cs221.group20.javafx'. The practice list stored within this class is a subset of the dictionary list.

Section 5.3 provides detail on significant algorithms.

6. FILES

6.1 Json files

The program stores data in a standard JSON file, formatted to the schema defined in the specification document SE.QA.CSRS DC3[2]. This is requested from the user at runtime, and then loading into the Application's internal data structures. When the program is closed, the file is automatically written to the location it was loaded from, providing a seamless experience for the user. This file is fundamental to the program's functionality.

The program requires a JSON file to load in the dictionary. The JSON package will request this file be loaded at runtime and all information will be saved at the end of the session. This file is fundamental to the program's functionality.

6.1 FXML files

The program uses FXML to store the layout and elements displayed on the Screen to fulfil each functional requirement in the original specification[2]. These files and the screen they display are listed below:

1. Add Word scene - addword.fxml
2. Dictionary scene - dictionary.fxml
3. Flashcard scene - flashcard.fxml
4. Practice List scene - practicelist.fxml
5. Match Meaning scene - matchthemeaning.fxml
6. Six Meaning scene - sixmeanings.fxml
7. Translation scene - translation.fxml

These files are stored in the resources folder in the package 'uk.ac.aber.cs221.group20', it is possible to add more scenes by including a new FXML file here and adding the scene in the SceneEnum Enumeration in the Class 'uk.ac.aber.cs221.group20.javafx.ScreenSwitch' in the format addWordScene("addword.fxml") for the Add Word scene above for example.

6.1 CSS files

A single CSS file was used to produce the program to the original specification[2], this was stored along with the FXML files in the resources folder in the package 'uk.ac.aber.cs221.group20' with the filename styles.css. This is used to add further styling to the JavaFX scenes outside of the FXML files.

7. INTERFACES

The program is written in Java 12, using a number of libraries.
Jackson version 2.9.4 is used as the json IO library.
JUnit Jupiter version 5.4.2 is used as the unit testing library.

JavaFX version 11 is used as the GUI library.

Library management is performed by Maven, if you wish to add more libraries or change the version of the library used, you will need to make the appropriate modifications to the pom.xml file stored at the root of the project. Maven will download the required libraries at compile time, this means library files will not otherwise be visible to the developer.

Json files are assumed to be formatted in the schema defined in 'SE.QA.CSRS DC3'[2].

/** What protocols should the maintainer be aware of? Maybe state that the program is written in Java. Mention JavaFX and JUnit. Basically, state what rules must be abided by for the program to work.

8. SUGGESTIONS FOR IMPROVEMENTS

If the program is to be improved, the program has four primary areas that it is recommended to implement in order to make the program better. These improvements would have been added to the program if given more time to develop it. The four suggestions for potential improvement are as described below:

8.1 Self Assessment Tests

A suggestion for improving the Self-Assessment Tests would be to increase the number of types of test that the user can do to add a greater variety of questions. Currently, the program includes the standard 'Translation', 'Six Meanings' and 'Match The Meanings' question types with support added for additional tests. An idea for one of these additional tests could include a 'spot the odd one out' tests on several dictionary definitions to find the definition and translation that is incorrect.

The implementation for adding extra questions would be relatively simple with any new tests extending the Question class and being added to the AssessmentGenerator class's generate assessment method so that they would be included in the list of questions that are generated. An additional FXML file and controller class would be needed so that the test has a GUI for the user to interact with the code required with the template for this being very similar to the existing self-assessment tests.

8.2 JUnit Tests

A suggestion for improving the JUnit tests would be to add more tests for each of the existing JUnit classes, 'JSONTest', 'DictionaryEntryTest', 'AssessmentGeneratorTest' and 'QuestionTest' to improve the general robustness of the system by testing for multiple possible scenarios. Examples could include adding additional JUnit tests into 'JSONTest' to see what happens when you try and load a file with fields missing.

As well as expanding the existing classes, additional tests could be added for the Controllers to test the robustness of the program's JavaFX within the 'test' package. These tests can be implemented using the TestFX library which specializes in testing JavaFX with these tests originally planned but were not implemented due to time constraints.

8.3 FXML

Another suggestion for improvement would be the programs FXML. In its current state, the program's pane cannot be decreased past a certain point due to some of the FXML not scaling properly leading it to over with the screens side bar. This was due to certain the different screens not being made consistently meaning some could scale well whilst others couldn't. This was planned to be fixed however it couldn't be done due to time constraints and would make the program more user friendly.

In order to implement these changes, it is recommended that most of the pages are redesigned within scene builder which has built in functionality to get panes to automatically scale relative to their parent.

8.4 Program's 'dictionary' Variable

One final suggestion for improving the program would be to add automatic alphabetical sorting to the program's 'dictionary' variable whenever a new variable is added to it. In its current state, the program is not resorted when a new definition is added to it, meaning the 'DictionaryController' has to sort the words alphabetically by its chosen language. Adding automatic resorting would mean that the controller no longer has to do the sorting manually, moving away functionality that isn't relevant to the class.

A way of implementing this could be done by implementing a compareTo method within the DictionaryEntry class. This method would be used to compare the alphabetical ordering of the words english or welsh definition depending on the current language ordering. With this method implemented, you could use a Collections.Sort method to resort the dictionary every time a new word is added.

9. THINGS TO WATCH FOR WHEN MAKING CHANGES

When making changes to the program there are several things that you must watch out for in order to avoid the system breaking. Possible issues deriving from changes could include errors at runtime such NullPointerExceptions or UI issues such as LoadExceptions. Listed below are various things to watch out for when making any changes to the system:

9.1 Changes to variables

When making changes to the program's variables including adding, removing or renaming variables, it is advised to take great care when doing so to avoid errors. Variables being changed should have their scope checked as well as their instances to see just how and where the variable is used in the program. It is important to fully refactor any changes to variables so that all instances of the variable are updated, with IDE's usually offering a way of doing this automatically. If changes are not fully refactored throughout the system, the program could fail to compile with other parts of the program to reference the old variable, throwing an ElementNotFoundException.

Changes to the variables type should also be watched closely, as whilst it is possible to refactor the type automatically through an IDE, it doesn't for instances where the variable is utilised in parts of the code. For example, the variable could be equated to a variable of a different type, leading to a TypeMismatchException.

9.2 Changes to FXML & Controllers

Alterations to the program's FXML & GUI such as adding or removing elements should also be taken with care as it is easy to get a LoadException. This is because every FXML file is linked to a Controller class, where each @FXML element in the Controller points to a element in the FXML file. Due to this, any elements that are removed from the FXML file with an fx:id should also be removed from the Controller as this could cause NullPointerExceptions.

Similarly, whenever a element is added that has an event is added e.g. 'onAction= #event', a corresponding event should be created within the FXML file's controller handling the event otherwise a LoadException will be thrown as it cannot find the event it refers to.

9.3 Changes to JSON file

When altering the format of the program's JSON file care should also be taken as the program's Jackson library deserializes the files by mapping the JSON directly onto the DictionaryEntry using the objects getters and setters. As a result of this, any additional fields added to the JSON file should also have matching getters and setter in the DictionaryEntry it is being deserialized to with matching names of the same type to avoid any errors.

In addition to these precautions, any variables added to the DictionaryEntry class will be automatically mapped onto the JSON file when saved, so it is important to mark them as ignored using @JSONIgnore if they aren't supposed to be mapped to the file, avoiding potential errors.

10. PHYSICAL LIMITATIONS OF PROGRAM

The program has some physical limitations that the user is required to meet in order to run properly. These limitations are as described below:

10.1 Screen Size

Limitations in the program's UI mean that users must have a screen size of at least 1100 X 680 in order to be able to see the program properly on their screens. This due to the UI struggling to scale below this set size, therefore the program's window is given this size as a minimum that cannot be made smaller.

10.2 Memory

The program's memory usage is dependent on the number of dictionary definitions that are loaded into the program as each definition is loaded into the program. The default dictionary.json file contains roughly 1300 definitions and uses around 200-300 MB of memory. As a result of this, it is advised to take note that the memory required will increase when more definitions are added.

11. REBUILDING & TESTING

11.1 How to Rebuild the project

Anyone looking to rebuild the program is recommended to open up the project within an IDE in order to gain an overview of the project. Once opened, find the 'Application' class which contains the main method, located within the package uk.ac.aber.cs221.group20.javafx. This class can be run by right clicking it on it with most IDE's selecting 'run Application.java', doing this will build the project.

11.2 How to test the project

Anyone looking to run the programs tests is also recommended to open the project within an IDE. Once doing so, look for the list of JUnit tests which are contained within the package uk.ac.aber.cs221.group20.test. This package will include 'AsssesemGeneratorTest', 'DictionaryEntryTest', 'JSONTest' and 'QuestionTest' and each test can be run by clicking the class and selecting 'Run XTest' where X is one of the four tests specified. After selecting run, the JUnit class will run with the result of each test being displayed with a green tick for a pass and a red cross for a fail.

REFERENCES

[1] Software Engineering Group Projects: General Documentation Standards. C. J. Price, N. W. Hardy, B.P. Tiddeman. SE.QA.03. 1.8 Release

[2] Software Engineering Group Projects: Welsh Vocabulary Tutor Requirements Specification. C. J. Price. SE.QA.CSRS 1.1 Release

DOCUMENT HISTORY

Version	*CCF No.*	*Date*	*Changes made to document*	*Changed by*
0.1	N/A	05/05/20	Initial document writeup	TP LW KB